Department of Computer Science
University of Bristol

COMS30030 - Image Processing and Computer Vision

Lab Sheet 01 - Part 1

Introduction to OpenCV Basics and Pixels

Lab Setup and Getting Started

Pui Anantrasirichai | n.anantrasirichai@bris.ac.uk
Original Slides: Tilo Burghardt | Majid Mirmehdi

What is OpenCV?

- OpenCV is a library framework for developing computer vision solutions. It is widely used in industry and research.
- It is (mainly) free for both academic and commercial use.
- It has C++, C, Python and Java interfaces and supports Linux, Windows, Mac OS, iOS and Android.
- OpenCV is designed for computational efficiency and with a strong focus on real-time applications.

Objectives of the First Lab Session

- Your first task is to setup OpenCV on the lab machines or your own machine. We use **Python** throughout all labs.

  https://github.com/opencv/opencv-python

- Then, start familiarising yourselves with the basics of OpenCV, such that after the lab you should be able to compile and run OpenCV programs, to create, draw into, load, save and display images, and to manipulate pixels.

- The following lab sheets will help you achieve this. Code and scripts for this lab are also available on the unit website.

- Then, get familiar with basic representation of the image: pixels, channels, and colours

Setting up OpenCV in the Lab MVB2.11

- First, open a terminal and create virtual environment:

```
$ python3 -m venv myproject
```

```
conda create -n myproject
```

- Activate your environment:

```
$ source myproject/bin/activate
```

```
conda activate pytorch-env
```

- Then it shows:

```
(myproject) [username@it0number ~]$
```

- Install OpenCV packages:

```
$ pip install --upgrade pip
$ pip install numpy opencv-python
```

```
conda install numpy
conda install -c menpo opencv
```

- Get out from the environment:

```
$ deactivate
```

```
conda deactivate
```

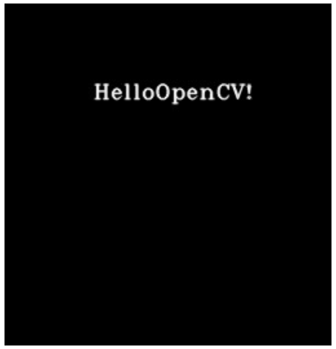
Setting up OpenCV in the Lab MVB2.11

- Now download the HelloOpenCV program hello.py from the course website.

A look inside the hello.py program...

hello.py

```
import cv2
import numpy as np

# create a black 256x256, 8bit, gray scale image in a matrix container
image = np.zeros([256,256], dtype=np.uint8)

# drawwhite text HelloOpenCV!
image = cv2.putText(image, "HelloOpenCV!", (70, 70), cv2.FONT_HERSHEY_COMPLEX_SMALL,
0.8, (255, 255, 255), 1, cv2.LINE_AA)

# save image to file
cv2.imwrite("myimage.jpg", image)

# construct a window for image display
namedWindow = 'Display window'

# visualise the loaded image in the window
cv2.imshow(namedWindow, image)

# wait for a key press until returning from the programe
cv2.waitKey(0)

# closing all open windows
cv2.destroyAllWindows()
```



Setting up OpenCV in the Lab MVB2.11

- Now try to load, display, create, draw and save the image
- Then, try to understand pixel representation in the image

Load and Display an Image

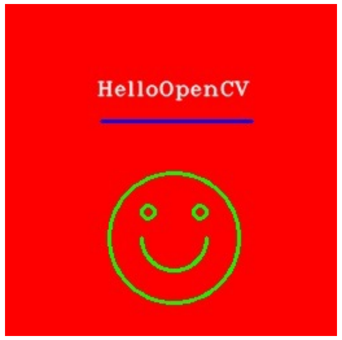
display.py

```python
import cv2
import numpy as np

# load image from a file into the container
image = cv2.imread("myimage.jpg", cv2.IMREAD_UNCHANGED)

# construct a window for image display
namedWindow = 'Display window'

# visualise the loaded image in the window
cv2.imshow(namedWindow, image)

# wait for a key press until returning from the programw
cv2.waitKey(0)

# closing all open windows
cv2.destroyAllWindows()
```

Create, Draw and Save

draw.py

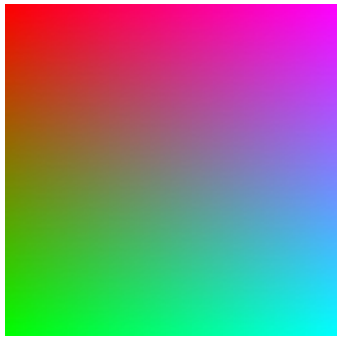
```
import cv2
import numpy as np

# create a red 256x256, 8bit, 3channel BGR image in a matrix container
image = np.zeros([256,256,3], dtype=np.uint8)
image[:,:,2] = 255

# put white text HelloOpenCV
image = cv2.putText(image, "HelloOpenCV", (70, 70),
cv2.FONT_HERSHEY_COMPLEX_SMALL, 0.8, (255, 255, 255), 1, cv2.LINE_AA)

# draw blue line under text
image = cv2.line(image, (74, 90), (190, 90), (255, 0, 0), 2)

# draw a green smile
image = cv2.ellipse(image, (130, 180), (25,25), 180, 180, 360, (0, 255, 0),
2)
image = cv2.circle(image, (130, 180), 50, (0, 255, 0), 2)
image = cv2.circle(image, (110, 160), 5, (0, 255, 0), 2)
image = cv2.circle(image, (150, 160), 5, (0, 255, 0), 2)

# save image to file
cv2.imwrite("myimage.jpg", image)
```



Access and Set Pixel Values

pixels.py

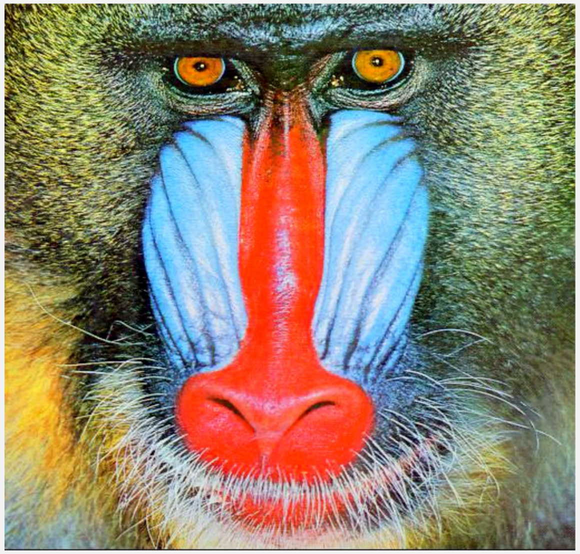
```
import cv2
import numpy as np

# create a red 256x256, 8bit, 3channel BGR image in a matrix container
image = np.zeros([256,256,3], dtype=np.uint8)

# set pixels to create colour pattern

for y in range(0, image.shape[0]):  # go through all rows (or scanlines)
        for x in range(0, image.shape[1]):  # go through all columns
                image[y, x, 0] = x # set blue component
                image[y, x, 1] = y # set green component
                image[y, x, 2] = 255 - image[y, x,1] # set red component

# construct a window for image display
namedWindow = 'Display window'

# visualise the loaded image in the window
cv2.imshow(namedWindow, image)

# wait for a key press until returning from the programe
cv2.waitKey(0)

# closing all open windows
cv2.destroyAllWindows()
```



Department of Computer Science
University of Bristol

COMS30030 - Image Processing and Computer Vision

www.ole.bris.ac.uk/bbcswebdav/courses/COMS30030_2019_TB-1/index.html



Lab Sheet 01 - Part 2

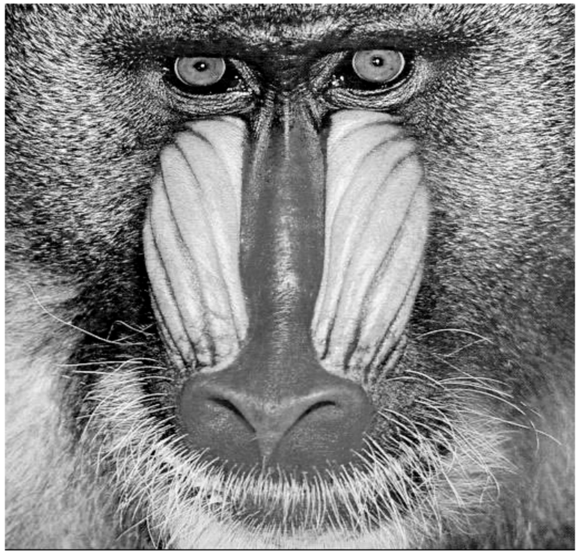
Introduction to OpenCV Basics and Pixels

Pixel Manipulation and Thresholding

Pui Anantrasirichai | n.anantrasirichai@bris.ac.uk
Original Slides: Tilo Burghardt | Majid Mirmehdi

Implementing Thresholding

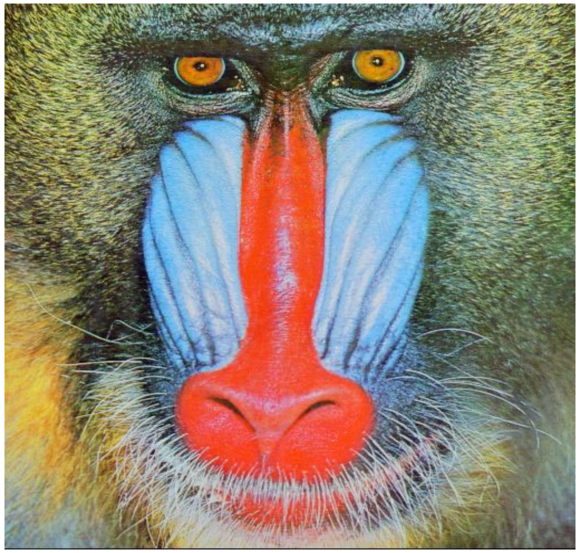
- Now that you are able to handle images, your next task is to write an OpenCV-based program that loads the mandrill.jpg greyscale image and that, pixel by pixel, sets all pixels above a certain value (maybe start with 128) to white (255) and all pixels equal or below the value to black (0).
- Experiment with different thresholding values and examine the resulting images. Can you highlight certain parts of the face (e.g. the nose, the eyes) with one or more specific thresholds?
- Compare your results to the output of the inbuilt OpenCV function `threshold`.



Optional: Thresholding Colour Images

- Whilst in greyscale images the brightness of a pixel is usually represented as a single byte in 2D array, colour images use three bytes to store information for one pixel, become 3D array. Bytes represent the BLUE, GREEN and RED channel in this order.

- Now, try to implement thresholding of the red, green and/or blue channels to highlight facial components in mandrillRGB.jpg, which now contains colour information.



- Sample answers are available at thr.cpp and colourthr.cpp if you are stuck.

- Also check the OpenCV function `inRange`.

First Steps in OpenCV: Basic Thresholding

thr.py

```
import cv2
import numpy as np

# Read image from file
image = cv2.imread("mandrill.jpg", 1)

# Convert to grey scale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY )

# Create mask from threshold
th = 128
gray_image = gray_image>128
gray_image = gray_image*255

# # Or threshold by looping through all pixels
# for y in range(0, image.shape[0]):  # go through all rows (or scanlines)
#       for x in range(0, image.shape[1]):  # go through all columns
#               if gray_image[y, x] > 128:
#                       gray_image[y, x] = 255
#               else:
#                       gray_image[y, x] = 0

# Save thresholded image
cv2.imwrite("thr.jpg", gray_image)
```

First Steps in OpenCV: RGB Thresholding

colourthr.py

```
import cv2
import numpy as np

# Read image from file
image = cv2.imread("mandrillRGB.jpg", 1)

# Threshold by looping through all pixels
th = 200
Blue = image[:,:,0]
image[:,:,0] = (Blue > th)*255
image[:,:,1] = (Blue > th)*255
image[:,:,2] = (Blue > th)*255

# # Or threshold by looping through all pixels
# for y in range(0, image.shape[0]):  # go through all rows (or scanlines)
#       for x in range(0, image.shape[1]):  # go through all columns
#               pixelBlue = image[y, x, 0]
#               pixelGreen = image[y, x, 1]
#               pixelRed = image[y, x, 2]
#               if (pixelBlue>200):
#                       image[y, x, 0] = 255
#                       image[y, x, 1] = 255
#                       image[y, x, 2] = 255
#               else:
#                       image[y, x, 0] = 0
#                       image[y, x, 1] = 0
#                       image[y, x, 2] = 0

# Save thresholded image
cv2.imwrite("colourthr.jpg", image)
```

Representation Basics – Pixels, Colours & Channels

Fun game: your task is to recover as much of the original mandrill colour image (2 channels, 8 bits per channel) from corrupted images.

1. View the corrupted images and, for each image, make a prediction of what has happened to it. Maybe look at individual colour channels or histograms to investigate your hypothesis.
2. Write a small OpenCV program for each image which can reconstruct the original mandrill colour image from it as well as possible.

Some images cannot be fully reconstructed.